

D3: Data-Driven Documents

作者：王琦

创建日期：2017 年 5 月 18 日

最后修改：2017 年 6 月 4 日

摘要：[D3.js](#) 是一款基于数据进行文档操作 (manipulating documents) 的 JavaScript 库。通过结合 HTML, SVG 及 CSS, D3 可以将数据栩栩如生地展现在用户眼前。

1. 基本图形

a) 阅读 `basic_shapes` 目录中的代码：`index.html` 及 `app/index.js`，猜想代码的运行效果。

b) 用浏览器打开 `index.html`，验证执行结果是否符合预期。

c) 修改 `index.js`，使用 JavaScript 在绘制结果中添加一条 path (可以尝试添加一个三角形)：

- [SVG Path](#)
- [SVG Paths](#)

d) 用 Webpack 完成模块解析，生成 `dist/bundle.js`。用浏览器打开页面，验证实验结果。

关于 Webpack 的内容可以参考：`day5/Webpack 2 -- JavaScript 构建工具.md`。

2. 基本操作：选择元素、事件监听及 DOM 操作

a) 阅读 `basic_operations` 目录中的代码，用浏览器打开 `index.html` 观察效果

b) 在 `basic_operations\app\index.js` 中添加代码，完成下述任务：

1. 将 circle 的半径变为原本的 1/2
2. 为 ellipse 增加事件监听，当点击 ellipse 时，将 circle 和 ellipse 的 边 的颜色变成红色
3. 为所有 rect 增加事件监听，当点击 circle 时，在控制台输出 `Circle clicked`
4. 针对类别 (class) 为 `log-rect` 的 circle，取消其在 3 中的事件监听，使得它每次被点击时输出它总共被点击的次数：
 - `log-rect` 第一次被点击时输出 `1`
 - `log-rect` 第二次被点击时输出 `2` ...
5. 针对 id 为 `color-rect` 的 circle，取消其在 3 中的事件监听，使得它每次被点击时，背景色在红和蓝之间跳变：
 - 背景初始化为白色

- 奇数次点击时背景变红 (red)
- 偶数次点击时背景变蓝 (blue)

3. 数据绑定

参考 [Thinking with Joins](#), 在此我们以这个 [不断变化的字符集](#) 为例, 每次 `update(data)` 被调用时都将发生:

- 新的字符 (`enter selection == data - elements`) 将以绿色的颜色渐入
- 已有的字符 (`update selection == data ∩ elements`) 将变为黑色
- 不再存在的字符 (`exit selection == elements - data`) 将以红色的颜色渐出

本章, 我们将根据 [D3 API](#), 以及一个例子来讲述 D3 的通用更新模式 (General Update Pattern).

3.1 初探 enter, update 和 exit

本节将通过解释 `general_update_pattern/1.html` (它采用了基本所有 Mike 的代码 —— [General Update Pattern I](#)), 来向读者介绍 D3 中 enter, update 和 exit 的用法。

```
// alphabet 是一个由 26 个字符组成的, 不再在后续的代码中变化的字母表
// 我们在 1.html 中不仅用 const 修饰它; 并且人为地保证不再对它进行 push 和 pop 操作
const alphabet = 'abcdefghijklmnopqrstuvwxyz'.split('');

// 每隔 1500 ms, 通过 d3.shuffle 生成一个含有 26 个乱序字符的字母表
// 选取前 n 个元素 (n 是一个落在 [0, 26] 区间内的随机整数), 并排序
// 所以, 每隔 1500 ms, update 函数都将接受到一个随机长度的排好序的字母表
d3.interval(function() {
  update(d3.shuffle(alphabet)
    .slice(0, Math.floor(Math.random() * 26))
    .sort());
}, 1500);
```

考察 `update(data)` 函数的定义:

```
const text = g.selectAll('text')
  .data(data);

// update selection 对应的元素将变黑
text.attr('class', 'update');

text.enter().append('text')
  // enter selection 对应的元素将变绿
  .attr('class', 'enter')
```

不难想象和验证, 在用浏览器打开 `1.html`, 完成首次调用 `update(alphabet)` 后, 浏览器中将出现

26 个绿色的字母 :)

现在我们来考虑更有趣的事 —— 更新这个字符串，使得：

- enter 元素展现为绿色
- update 元素展现为黑色
- exit 元素直接消失

假设原字符串为 s ，新字符串为 t 。在此需要强调的是，**entering 元素（新出现的元素）并不指代 t 与 s 的差集（集合 $t - \text{集合 } s$ ）——亦即在 s 中不出现，而在 t 中出现的字符。**我们形式化地描述本例中的 enter, update 和 exit 元素对应的数据（字符）：

- `enter characters = t.length > s.length ? t.substring(s.length) : ''`. 即字符串 t 长于字符串 s 的那部分子串。[点击此处查看 substring API](#)
 - 观察页面，不难发现新字符串 t 中，**总是只有后面那部分下标大于 $s.length$ 的字符是绿色的。**
- `update characters = t.substring(0, s.length)`. 即字符串 t 中以 $s.length$ 为长度的子串 `t.substring(0, s.length)`. 当 s 的长度大于 t 的长度时，update 元素为 t 本身。
 - **t 中，总是只有前面那部分下标小于 $s.length$ 的字符是黑色的。**
- `exit characters = s.substring(t.length)`.
 - **s 中，总是只有后面那部分下标大于 $t.length$ 的字符会消失。**

该描述是根据 [D3 API -- selection.data\(\[data\[, key\]\]\)](#) 推得的。在没有指明 `key function` 的情况下，新数据 (data) 与旧元素 (element) 根据下标，一一对应。

If a key function is not specified, then the first datum in data is assigned to the first selected element, the second datum to the second selected element, and so on.

enter, update, exit 部分以及新旧数据元素的对应关系是本小节最重要的内容。请同学们阅读并理解 API 定义和代码实现，确信自己能独立推导出上述内容。

3.2 Key Function

本小节将考察例子 `general_update_pattern/2.html`（它采用了基本所有 Mike 的代码 —— [General Update Pattern II](#)），来向读者介绍如何在 d3-selection 中显式指定元素的键，进而改变 `data-join` 时产生的 enter, update 和 exit 元素。

```

const update = data => {
  // DATA JOIN
  // Join new data with old elements, if any.
  const text = g.selectAll("text")
    .data(data, d => d); // 1

  // UPDATE
  // Update old elements as needed.
  text.attr("class", "update");

  // ENTER
  // Create new elements as needed.
  //
  // ENTER + UPDATE
  // After merging the entered elements with the update selection,
  // apply operations to both.
  text.enter().append("text")
    .attr("class", "enter")
    .attr("dy", ".35em")
    .text(d => d) // 2
    .merge(text)
    .attr("x", (d, i) => i * 32); // 3

  // EXIT
  // Remove old elements as needed.
  text.exit().remove();
}

```

同样的，[D3 API -- selection.data\(\[data\[, key\]\]\)](#) 详尽介绍了 `key function` 的用法。请阅读官方文档，回答关于上述代码片段的问题：

- 为什么标记 1 处需要这个 `key function`，它使得本程序和 `1.html` 中的程序有何不同？（具体是通过一个匿名函数 `d => d` 实现的，但这个实现与本问题无关）
- 相比 `1.html`，`2.html` 的 `.text(d => d)` 从原本的标记 3 替换到了标记 2 处
 - 若放在标记 3 处执行，可以达到相同的显示效果吗？
 - 为什么 `2.html` 中要将它放在标记 2 处？
- 相比 `1.html`，`2.html` 的 `.attr("x", (d, i) => i * 32)` 从原本的标记 2 替换到了标记 3 处。为什么要这样做？若将其放回标记 2 处，能达到相同的显示效果吗？

3.3 添加动画效果

本章的最后一节，我们将考察 `general_update_pattern/3.html`，亦即最初给出的 [不断变化的字符集 \(General Update Pattern III\)](#)。每次 `update(data)` 被调用时都将发生：

- 新的字符 ($\text{enter selection} == \text{data} - \text{elements}$) 将以绿色的颜色渐入
- 已有的字符 ($\text{update selection} == \text{data} \cap \text{elements}$) 将变为黑色
- 不再存在的字符 ($\text{exit selection} == \text{elements} - \text{data}$) 将以红色的颜色渐出

通过阅读上述代码和 [d3-transition API](#), 相信你可以理解在 D3 中, 动画效果是如何实现的。